CSE 510
Web Data Engineering

Java Servlets
Install and Check Tomcat
Installing Tomcat

• Install stable production release
  – We will be using Apache Tomcat version 6.0.20
  – Do not install alpha, beta, milestone or nightly builds

• You need Java SE Development Kit (JDK) 6

• On Windows, use the self-extracting .exe and follow directions
Starting and Testing Tomcat

• Start Tomcat using bin/startup.bat or “Start Tomcat” icon in program group
  – Preferably, do not set up Tomcat as an “automatic start” service

• You can also use a Tomcat Launcher Plug-in within Eclipse

• Open [http://localhost:8080/](http://localhost:8080/)
  – You should see Jakarta project home page

• Open [http://localhost:8080/examples/jsp/dates/date.jsp](http://localhost:8080/examples/jsp/dates/date.jsp)
HTTP Requests and Responses
HTTP Basics

• TCP/IP protocol used by web servers
• Synchronous
  – Client sending request waits for response
• Stateless
  – All info needed by server-side must be contained in HTTP request
  – Using appropriate session management techniques, we can go around restrictions of statelessness
• We show next the request and response message strings that go back and forth in interactions
  – Only for educational purposes
  – You will never code such strings directly
  – The application server will do it for you
Syntax of an HTTP Request

• <method> <request URI> <HTTP-version>
  – Important ones: GET & POST
  – See Table 3.1 of textbook for explanations of other methods: HEAD, PUT, DELETE, CONNECT, OPTIONS, TRACE

• Header fields
  – Accept: text/html, text/xml, ...
    (acceptable response types)

• Message body (optional) (after blank line)
Example

GET / HTTP/1.1
Host: db.cse.buffalo.edu
User Agent: IE/5.0
Accept: text/html, text/xml

...
Syntax of an HTTP Response

- `<HTTP-version> <status-code> <reason>`
  - For example, status codes from 500-599 indicate server-side errors
  - See Table 3.2 for typical HTTP response codes
- Header fields
  - `Content-Type: text/html (or other type)`
- Message body (optional) (after blank line)
Communicating Data Provided in Forms: GET, POST and Parameters

- Consider the multiplication page

```html
<html>
  <head><title>Multiplier Form</title></head>
  <body>
    Welcome to the page that multiplies by 3
    
    <form method="GET" action="multiply">
      Provide the number to be multiplied:
      
      <input type="text" name="num"/>
      <input type="submit" value="Click to Submit"/>
    </form>
  </body>
</html>
```
When and How to Use POST (Instead of GET)

- Upon submitting “3” the browser emits URL
  
  http://localhost:8080/multiplier/multiply?num=4

  GET /multiplier/multiply?num=4 HTTP/1.1
  Host: localhost:8080

- If your HTML form may create more than 255 characters use `<form method="POST" ...`
  
  - Form data will be in body of http request
  
  POST /multiplier/multiply HTTP/1.1
  Host: localhost:8080

  num=4
More Input Forms: Dropdown Menus
More Input Forms: Checkboxes
Encoding URIs

- HTTP only permits letters, digits, underscores and a few more
- Browsers take care of “special” symbols, using the RFC2277 encoding
Example of Encoding Characters in a URI Using the RFC2277

• Consider a page asking for emails
  
  ```html
  <html>
    <head><title>Email Submit Page</title></head><body>
      <form method="GET"
          action="http://localhost:8080/subemail.jsp">
        Type your e-mail here:
        <input type="text" name="eml"/>
        <p>
        <input type="submit" value="Click Here"/>
      </form>
  </body></html>
  ```

• User types mpetropopo@buffalo.edu
  GET /subemail.jsp?eml=mpetropo%40buffalo.edu HTTP/1.1
  Host: localhost:8080
Some Useful Aspects of HTTP

- URI redirection
- Refresh
  - Instruct the browser to reload every N seconds
  - `<meta http-equiv="refresh" content="300">`
  - Refresh: 300
Servlets: The 101 of Java-based Web Server-Side Programming
Java-Based Server-Side Programming 101: Servlets

- Servlet: Java program run inside the app server (Tomcat)
- Inputs HTTP requests
  - App server provides them in appropriate object format
- Typically (but not necessarily) return HTTP responses of HTML content type
Multiplication Form and Servlet: The HTML Form Gets Input, Calls Servlet

- Create Web app (directory) `multiplier` under `webapps`
- Place `multiplier.html` in it
- Browse to http://localhost:8080/multiplier/multiplier.html
- When form is submitted, browser issues HTTP GET request
  - ACTION specifies URL to be invoked
  - URL of servlet may be relative, as in `multiplier.html`, or absolute:
    http://localhost:8080/multiplier/multiply
import java.io.*;
/* following packages encapsulate Servlet API */
import javax.servlet.*;
import javax.servlet.http.*;

public class MyMultiplier extends HttpServlet {
    /* Overrides doGet coming with HttpServlet */
    public void doGet(HttpServletRequest req,
            HttpServletResponse res)
            throws ServletException, IOException {
        // Method implementation...
    }
}
res.setContentType("text/html");
/* By having set content to text/html */
/* PrintWriter encodes accordingly */
PrintWriter out = res.getWriter();
out.println("<html><head><title>Multiply by " + 3
  + "</title></head><body>");
String parameter = req.getParameter("num");
/* Ignoring the possibility that parameter is not
   integer */
out.println(parameter + " * " + 3 + " = " +
  3 * (Integer.parseInt(parameter)));
out.println("</body></html>");
}
Compiling & Deploying the Servlet

• **Place** `MyMultiplier.java` in `multiplier/src`
  – Not necessary, but good principle to separate java sources from classes

• **Compile** `MyMultiplier.java`
  – Include in CLASSPATH environment variable
    `<CATALINA_HOME>/lib/servlet-api.jar`

• **Place** `MyMultiplier.class` in `multiplier/WEB-INF/classes`
Deployment Descriptor & URL Mapping

- Map the servlet class to a URL pattern in the deployment descriptor `multiplier/WEB-INF/web.xml`

```xml
<web-app>
  <servlet>
    <servlet-name>multiplier</servlet-name>
    <servlet-class>MyMultiplier</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>multiplier</servlet-name>
    <url-pattern>/multiply</url-pattern>
  </servlet-mapping>
</web-app>
```

- After restarting Tomcat, you can access servlet at

```
http://localhost:8080/multiplier/multiply?num=4
```
• URL pattern may include *(wildcard)*

```xml
<servlet-mapping>
  <servlet-name>action</servlet-name>
  <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

• Any URL pattern matching *.*.do will invoke the action servlet

• We’ll see this again in Struts implementations (indeed this example is from Struts)
Servlet Life Cycle

- First time a servlet is called:
  - `init()` method is invoked
    - Normally provided by `HttpServlet`
    - Unless you want to set up resources that exist for the whole lifetime of the servlet (rare)
    - Object (extending `HttpServlet`) is instantiated and becomes memory resident from now on
      - Class variables exist for entire life of object
  - Series of GET, POST, ... HTTP calls lead to `doGet()`, `doPost()`, etc method invocations
  - Servlet removed with `destroy()`
    - Tomcat may call `destroy()` any time
    - You may write your own `destroy()` to save state
Handling POST Method Calls

- Whether parameters are communicated by GET or POST is normally irrelevant to your code
- However, you have to provide (override) `doPost()` of `HttpServlet`

```java
public void doPost(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {
    doGet(req, res);
}
```
Handling Other Method Calls

- DELETE, HEAD, OPTIONS, PUT, TRACE
- Corresponding `doDelete()`, `doHead()`, etc
- Normally developer does nothing
- `HttpServlet` provides defaults
Servlet Initialization Parameters: Definition in web.xml

• Assume we want to change the multiplication factor without having to change and recompile the MyMultiplier.java servlet

• Add initialization parameter in web.xml

```xml
<web-app>
  <servlet>
    <!-- … servlet stuff we’ve seen … -->
    <init-param>
      <param-name>times</param-name>
      <param-value>5.0</param-value>
    </init-param>
  </servlet>
</web-app>
```
• Access to initialization parameters by invoking
  `getInitParameter`

  ```java
  String timesStr = getInitParameter("times");
  ```
Servlet Context Path

- Default context name of Web application is the name of the webapps subdirectory
  - In running example, multiplier
- Create alias context name if you want to hide the subdirectory name or effect non-default actions on your application’s servlets
- Add Context element in conf/server.xml, inside <Host name="localhost" ...>
  - <Context path="/mult" docbase="multiplier"/>
- Path is matched against URLs’ beginning
  - Must be unique
  - Try http://localhost:8080/mult/multiply?num=4
Automatic Reload

- Default configuration does not check whether class files are replaced
  - Appropriate setting in production mode
- We can avoid stopping and restarting Tomcat during development/compilation by enabling automatic reloading of servlet class files
  - For an individual web application, add file `context.xml` under `<WEBAPP_HOME>/META-INF/` and just add
    ```xml
    <Context path="" reloadable="true">
    ```
  - To effect automatic reload for all applications add, edit file `<CATALINA_HOME>/conf/context.xml`, and add `reloadable="true"` attribute to the `Context` element
What is Wrong with Servlets

- The “look” of the resulting HTML is buried in println() statements
- Web designers cannot work this way
- Business logic and presentation horribly mixed
- other issues...
Some Additional Items for Your “To Do” List

- Automatic Reloading of Servlets
- **Deploy and modify the programs we’ve seen**